Chapter 8: Memory Management

<Summary>

• Various memory management algorithms differ in many aspects

- Contiguous allocation, Paging, Segmentation

• Hardware support

- A simple base-limit register pair is enough for contiguous allocation
- A mapping table is required for paging and segmentation

• Performance

- TLB can reduce the performance degradation to an acceptable level

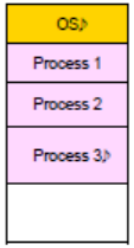
• Fragmentation

- Internal fragmentation: paging
- External fragmentation: contiguous allocation, segmentation
- One solution to the external fragmentation is compaction

• Relocation, swapping, sharing, and protection is another considerable issue in memory management

1. Background

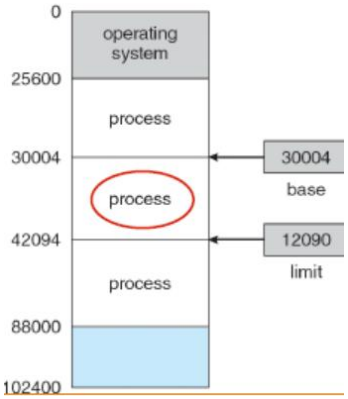
• As a result of CPU scheduling, we can improve both
  - the utilization of the CPU and
  - the speed of the computer's response to users.



• To realize this increase in performance
  - We must keep several processes in memory

• Program must be brought into memory and placed within processes for it to be run
  - Each process has a separate memory space.
  - Base register and limit register are used to determine the process address space in physical memory.
  - Base register holds the smallest legal physical memory address
  - Limit register holds the size of the process address space.

2. A base and a limit register



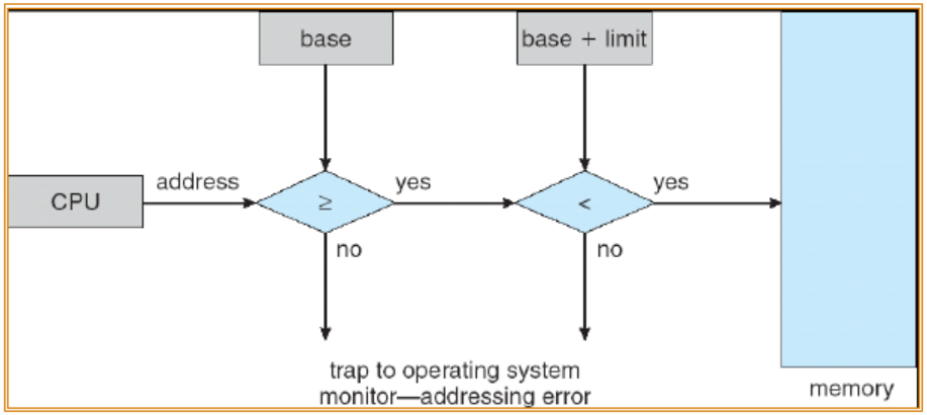
• the second process in the left memory map

• the base register: 30004 <- starting point
• the limit register: 12090
• the range in physical memory this process can access is from 30004 through 42094 (inclusive)

3. H/W address protection with base and limit registers

• Protection of memory space is accomplished by having the CPU compare every address generated in user mode with the registers (base and limit registers).

• Every illegal memory access results in trap to the OS.



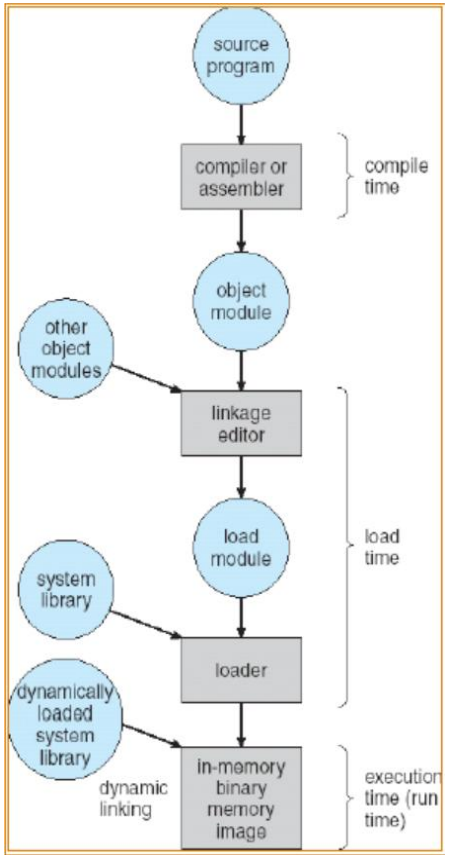
4. Input queue

• A program resides on a disk as a binary executable file.

- To be executed, the program must be brought into memory and placed within processes.
- Depending on the memory management in use, the process may be moved between disk and memory during its execution.
- The processes on the disk are waiting to be brought into memory for execution from the input queue.

• Input queue – collection of processes on the disk that are waiting to be brought into memory for execution

5. Multi-step Processing of a User Program



• User programs go through several steps before being run

• Addresses may be represented in different ways during these steps.
  - Instructions
  - Data

• Addresses in the source program are generally symbolic. (int A;)

• A compiler will typically bind these symbolic address to relocatable addresses

• Linkage editor or **loader** will bind the relocatable addresses to absolute addresses.

6. Address Binding of Instructions and Data to Memory

일반적으로, 프로그램은 디스크에서 바이너리 파일로 존재한다. 프로그램을 실행하기 위해서는 디스크에서 메모리로 읽어와야 한다. OS가 메모리를 어떻게 관리하느냐에 따라 프로세스는 메모리에서 디스크로 이동하거나, 다시 메모리로 읽히기도 한다. 실행하기 위해 디스크에서 메모리로 읽히기를 기다리고 있는 프로세스들이 모여 입력큐 input queue가 형성된다.

Address binding of instructions and data to memory addresses can happen at three different stages

1) Compile time: 컴파일 할 때 프로세스가 메모리의 어떤 위치에 존재할 지 알고있다면, absolute code can be generated; must recompile code if starting location changes

2) Load time: 컴파일러가 relocatable code를 생성한다. if memory location is not known at compile time

3) Execution time: Binding delayed until run time if the process can be moved during its execution from one memory segment to another.
  (ex) swapping out -> swapping in
  - Need hardware support for address maps
  (e.g., base and limit registers).

7. Logical vs. Physical Address Space

• The concept of a logical address space that is bound to a separate physical address space is central to proper memory management

1) Logical address
  – generated by the CPU, also referred to as virtual address

2) Physical address – address seen by the memory unit

• Logical and physical addresses are the same in compile time scheme;

• “ differ in execution-time address-binding and load-time address binding schemes

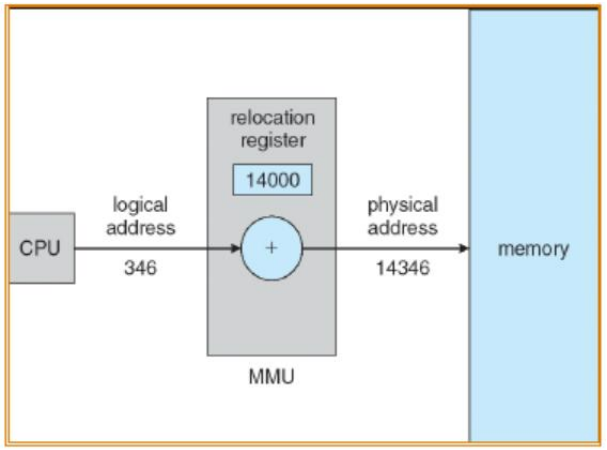
8. Memory-Management Unit (MMU)

• Hardware device that maps virtual (logical) to physical address

• In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory

• 사용자 프로그램은 논리적인 주소를 다루지, 물리적인 주소를 다루지 않는다.

9. Dynamic relocation using a relocation register



• base register = relocation register

• The value in the relocation register is added to every address generated by a user process at the time it is sent to memory.

10. Swapping

프로세스가 실행되려면 메모리에 있어야 한다. 그러나 프로세스는 임시로 메모리를 벗어나 backing store로 swapping 될 수 있다. 이는 나중에 실행을 계속하기 위해 다시 메모리로 불러들인다.

• A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution

- Swap in, swap out

• Backing store

- fast disk large enough to accommodate copies of all memory images for all users
- must provide direct access to these memory images

• Roll out, roll in

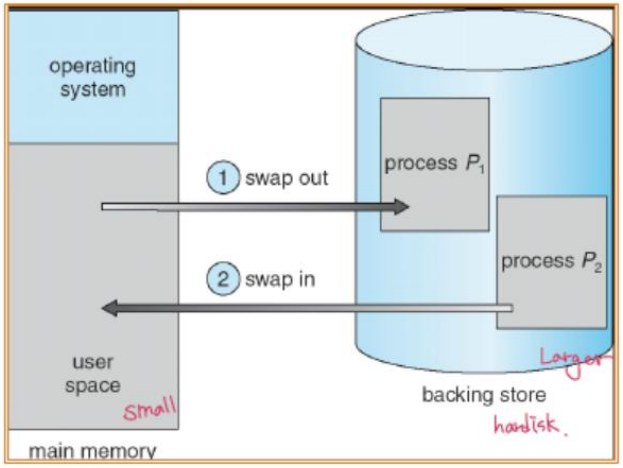
- swapping variant used for priority-based scheduling algorithms
- lower-priority process is swapped out so higher-priority process can be loaded and executed

• Major part of swap time is transfer time;

• total transfer time is directly proportional to the amount of memory Swapped 전송시간 ∝ swapping된 메모리 양

• Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

11. Schematic View of Swapping



• Dispatcher swaps out a process to input queue in the backing store when the memory space is not enough

• Dispatcher swaps in a process from the backing store to ready queue

12. Contiguous Allocation

• Main memory is usually divided into two partitions:

1) Resident operating system, usually held in low memory with interrupt vector

2) User processes then held in high memory

• Contiguous Allocation

- Each process is contained in a single contiguous section of memory

- Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data

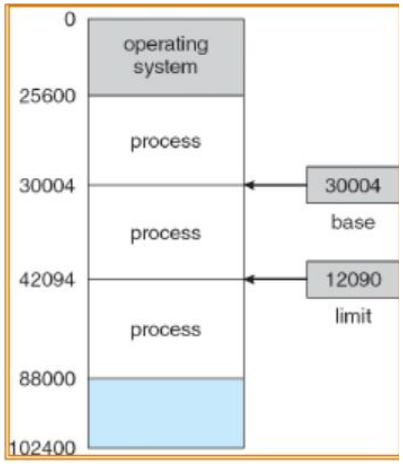
1) Relocation register contains value of the smallest physical address;

: 메모리 보호를 제공하기 위한 레지스터, 물리적 주소의 최소값 제공, 운영체제의 크기를 동적으로 변화시킬 수 있다.

2) Limit register contains range of logical addresses

: 논리적 주소는 Limit register보다 작아야 한다.

12-1. Example of contiguous allocation



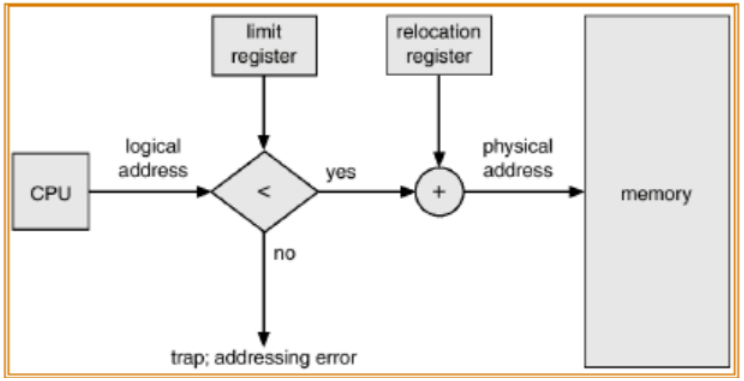
• Several user processes in memory at the same time

• Each process in a contiguous section of memory

12-2. Protection with limit and relocation register

• Each logical address must be less than the limit register.

• The MMU maps the logical address dynamically by adding the value in the relocation register, which is sent to memory.

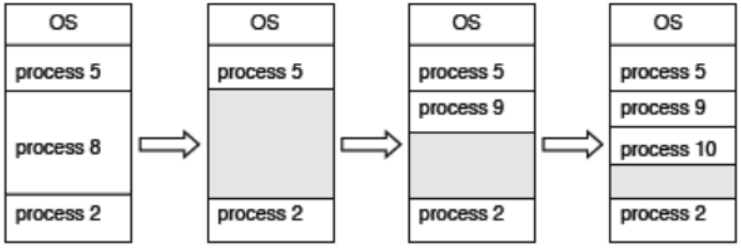


Contiguous Allocation (Cont.)

• Hole – a block of available memory;
holes of various size are scattered throughout memory

• Operating system maintains information about:
a) allocated partitions b) free partitions (hole)

• When a process arrives, it is allocated memory from a hole large enough to accommodate it



13. Dynamic Storage-Allocation Problem
: How to satisfy a request of size n from a list of free holes

• First-fit: Allocate the first hole that is big enough
• Best-fit: Allocate the smallest hole that is big enough;
  - must search entire list, unless ordered by size
  - produces the smallest leftover hole
• Worst-fit: Allocate the largest hole;
  - must also search entire list
  - produces the largest leftover hole

First-fit and best-fit better than worst-fit in terms of speed and storage utilization

14. Fragmentation

내부 단편화 : 분할된 영역 > 작업크기 <- 페이징
외부 단편회 : 분할된 영역 < 작업크기 <- 세그멘테이션

• External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous

메모리는 남지만 조각으로 떨어져 있어 할당이 불가능한 상태

• Internal Fragmentation – allocated memory may be slightly larger than requested memory;

- 이러한 차이로 발생하는 공간은 파티션 내부에 속해 있지만 사용되 않는다.

• Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible only if relocation is dynamic, and is done at execution time

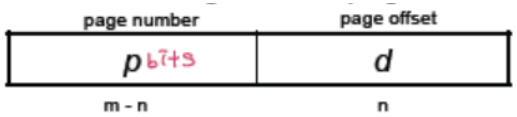
15. Paging – a memory management scheme
• allows that
  - Physical address space of a process can be noncontiguous
  - Process is allocated physical memory whenever the latter is available
• Divide physical memory into fixed-sized blocks called frames
  {size is power of 2, between 512 (=29) bytes and 16384(=214) bytes}
• Divide logical memory into blocks of same size called pages
• Keep track of all free frames
• To run a program of size n pages,
  need to find n free frames and load program

• Set up a page table to translate logical to physical addresses
• No external fragmentation but Internal fragmentation

16. Address Translation Scheme
• CPU가 사용하는 논리적 주소는 다음과 같이 나누어진다

1) Page number (p) – used as an **index** into a page table which contains base address of each frame in physical memory
2) Page offset (d) – 페이지 안의 offset, combined with base address to define the physical memory address that is sent to the memory unit

• 논리적 주소 공간이 2^m, 페이지 크기가 2^n이라면

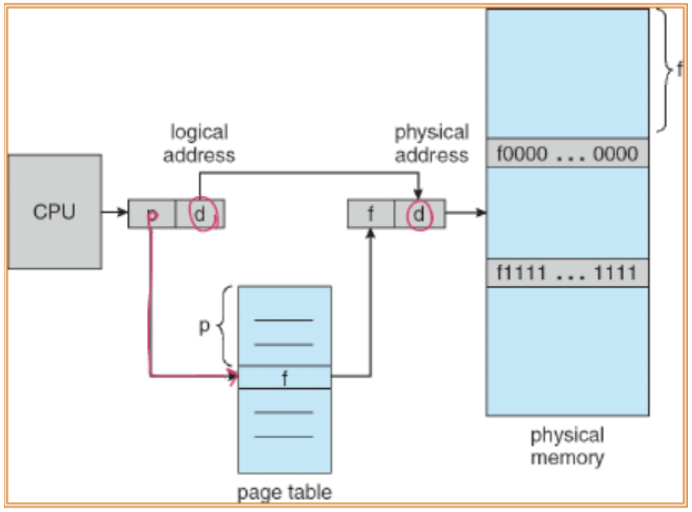


- The high-order m - n bits of a logical address designates the page number
- The n low-order bits designate the page offset

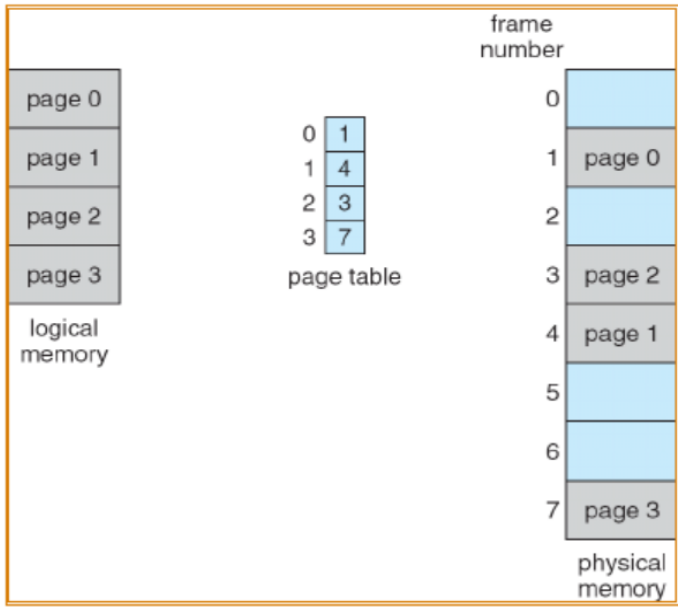
17. Address Translation Architecture

• 페이지 테이블은 물리적 메모리 안에 있는 각각의 페이지의 base 주소를 담고있다.

• A page table is created for each process.



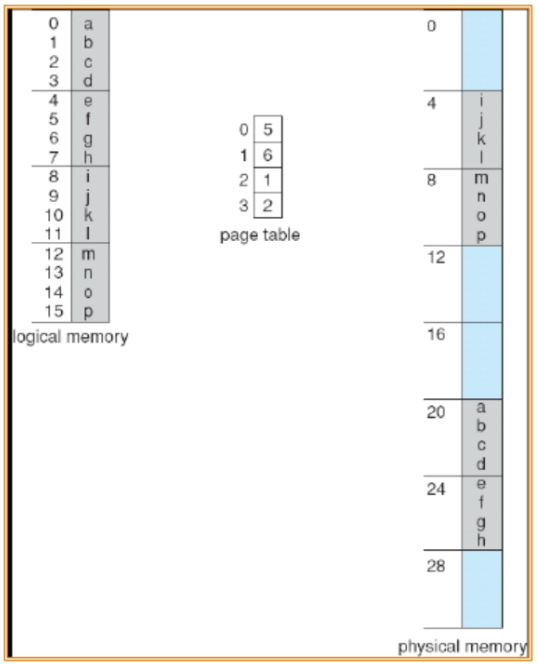
15-1. Paging Example



- Page (Frame) size: 4 bytes
- Physical memory: 32 bytes
  - 8 frames
  - 5 bits for addressing
  - 3 bits for page table entry
- Logical address space: 16 bytes
  - 4 pages
  - 4 bits for addressing

- Logical address 4 bits
  - Higher 2 bits for page number
  - Lower 2 bits for page offset

- Logical address 0 (0000) is
  - page 0, offset 0 → 5 x 4 + 0 = 20
- Logical address 1011 (1011) is
  - Page 2, offset 3 → 1 x 4 + 3 = 7



16. Page Table Size

• Page table size depends on

- The size of a logical address space of a process and a pagetable entry.

• Usually, each page-table entry is 4 bytes long,

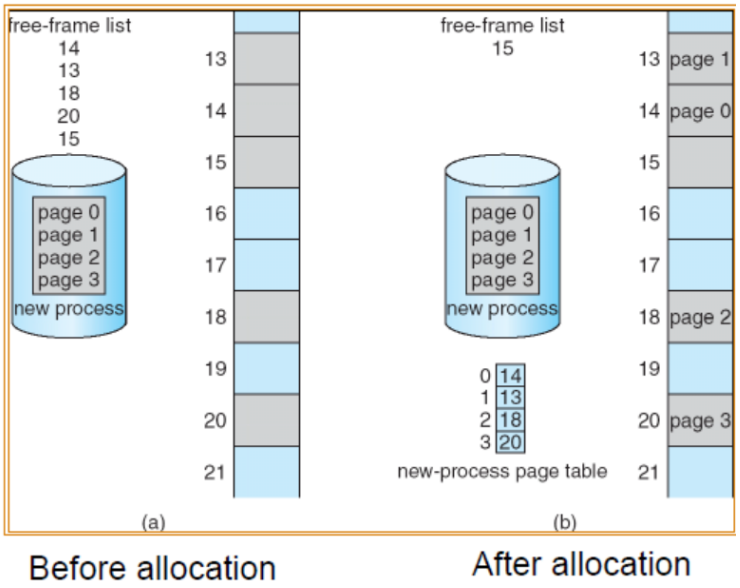
- but that size can vary as well.

• A 32-bit entry can point to one of 2^32 physical page frames.

• If a frame size is 4 KB (=2^12)

- A system with 4-byte page table entries can address 244 bytes (16 TB) of physical memory

17. Free Frames



Before allocation

After allocation

• When a new process created with
- 4 pages of address space

• OS should check that
- 4 frames are available
• A new page table created
- page 0 frame 14
- page 1 frame 13
- page 2 frame 18
- page 3 frame 20

18. Implementation of Page Table

• Page table은 메인 메모리에 저장하고, 2개의 레지스터가 사용된다
- Page-table base register (PTBR) points to the page table
- Page-table length register (PRLR) indicates size of the page table

• 이 기법의 문제점은, 명령어와 데이터에 접근할 때마다 메모리 접근을 2번 해야 한다는 것이다
- One for the page table and 페이지 테이블 접근 한번
- One for the data/instruction 명령어 또는 데이터에 접근

• The two memory access problem(메모리에 두번씩 접근하는 단점) can be solved by the use of a special fast-lookup hardware cache called translation look-aside buffers (TLBs)

19. Translation look-aside buffer (TLB)

• is an associative, high-speed memory
- A cache for the page table

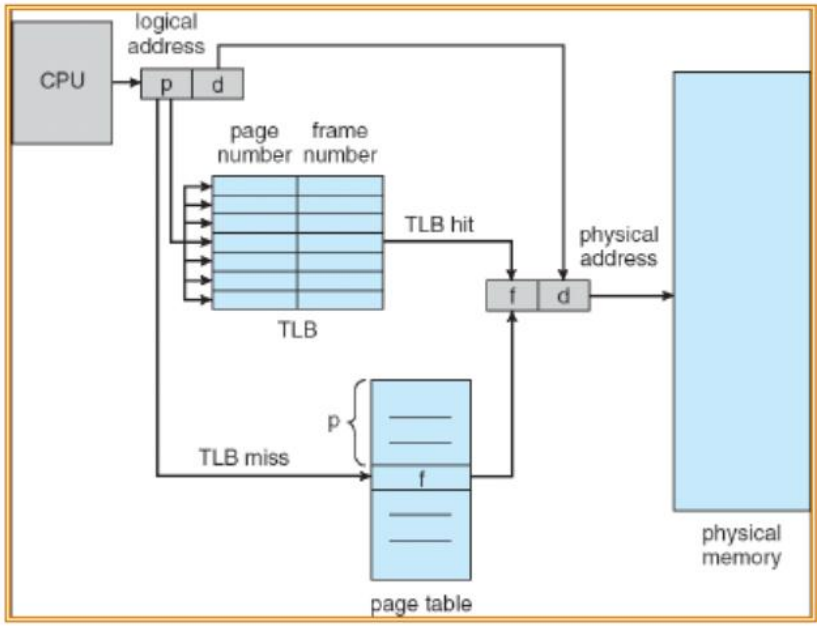
• TLB contains only a few of the page-table entries
- When a logical address is generated by the CPU, its page number is presented to the TLB.

논리적인 메모리를 물리적인 메모리로 변환하 때 TLB를 먼저 검색한다. i) TLB에 페이지 번호가 발견되면 즉시 변환한다.

ii) 없으면 페이지 테이블을 검색한다

- If the page number is found (TLB hit), its frame number is immediately available and is used to access memory
- If the page number is not in the TLB (TLB miss), a memory reference to the page table must be made
- If frame number is obtained, we can use it to access memory
- We add the page number and frame number to the TLB
- If the TLB is already full of entries, OS selects one for replacement
• replacement policy: least recently used (LRU), random

19-1. Paging Hardware With TLB



20. Effective Access Time

TLB 접근 20ns, 메모리 접근 100ns 걸린다면

페이지 번호가 TLB에 있으면, 메모리 접근에 20 + 100 = 120ns 걸림

그렇지 않으면 메모리에 2번 접근해야 하므로 20+100+100=220ns걸림

유효한 메모리 접근 시간을 계산하려면

Effective Access Time = 120* α + 220(1- α)

여기서 α는 주소를 변환할 때, TLB에 페이지 번호가 존재할 확률이다 (Hit ratio)

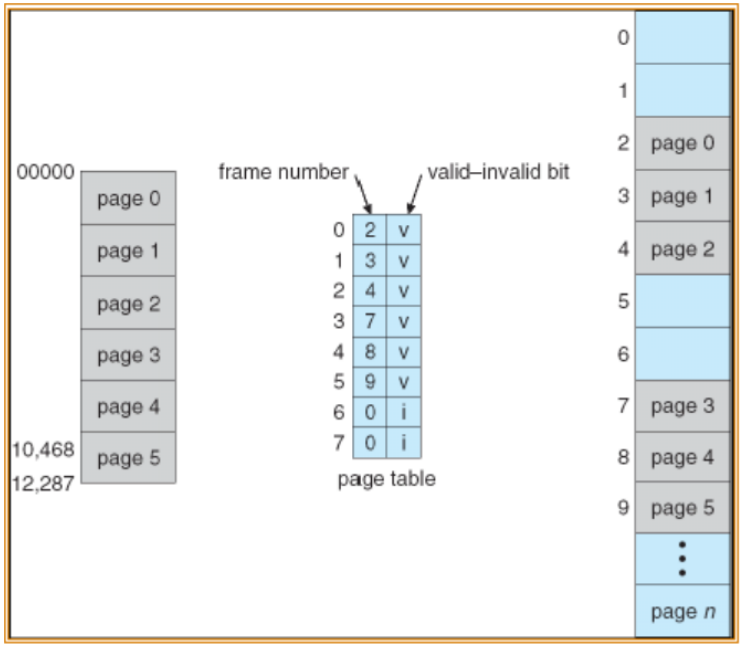
- TLB Lookup time = ∑ time unit
- Memory access time = τ time unit

21. Memory Protection

• Memory protection implemented by associating protection bit with each frame
  - One bit can define a page to be read-only, or read-write
  - An attempt to write to a read-only page causes a hardware trap to the operating system.
• Valid-invalid bit is attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "invalid" indicates that the page is not in the process' logical address space

22. Valid (v) or Invalid (i) bit in a page table

• Suppose that in a system with
  - 14-bit address space,
  - 2KB of page (frame) size
• A process has 6 x 2KB of address space.
  - Page table size: 8 (=2^3)
• Addresses in pages 1,2,3,4,5 are mapped normally through the page table
• Any attempt to address in pages 6 or 7, will find that the valid-invalid bit is set to invalid, and the trap is generated



Page Table Structure

• Hierarchical Paging
• Hashed Page Tables
• Inverted Page Tables

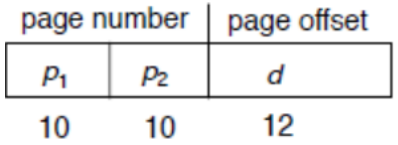
1. Hierarchical Page Tables

• Consider a system with
- a 32-bit logical address space, 4 KB of page size
• Required page table size
- (2^32 / 2^12) x 4 byte = 4 Mbytes too big for a page table.
• Solution to this problem:
• Break up the logical address space into multiple page tables
• A simple technique is a two-level page table

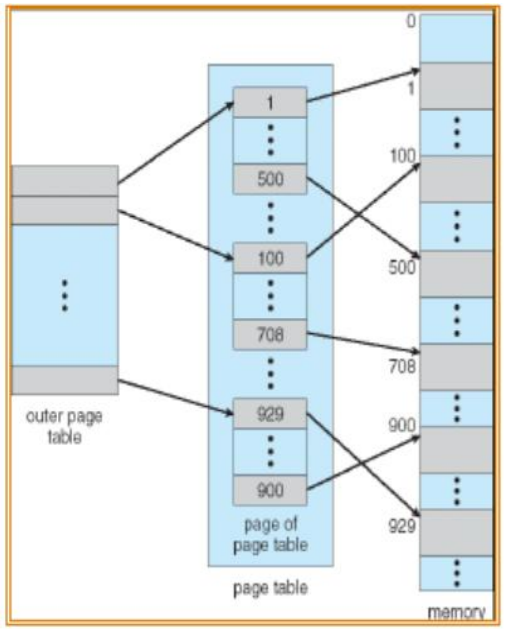
2. Two-Level Paging Example

• A logical address (on 32-bit machine with 4KB page size) is divided into:
- a page number consisting of 20 bits
- a page offset consisting of 12 bits (4K = 2^12bits)
• Since the page table is paged, the page number is further divided into:
- a 10-bit page number
- a 10-bit page offset
• Thus, a logical address is as follows:
where p1 is an index into the outer page table, and p2 is the displacement within the page of the outer page table

p1은 상위 페이지테이블 인덱스고 p2가 상위페이지테이블 인덱스?



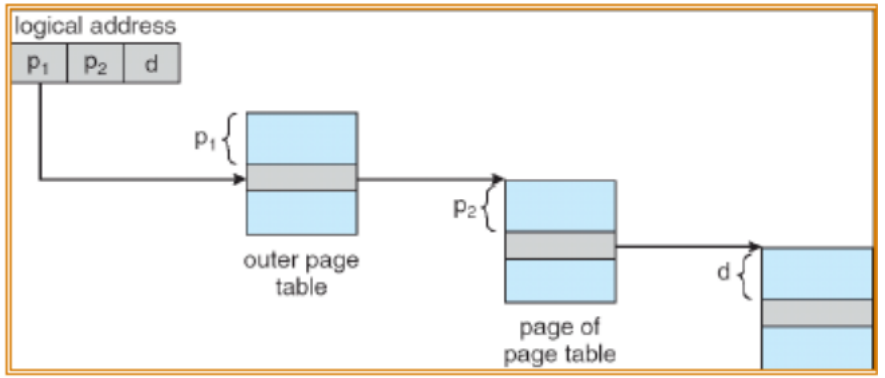
3. Two-Level Page-Table Scheme



• Logical address
- 32-bit address with 4KB page
• # of page entries
- 2^32 / 2^14 = 2^20
• 20 bits for page number
- 10 bits for index for outer page table
- 10 bits for index for the subpage table
• # of entries in outer page table
- 2^10
• # of entries in each sub-page table
- 2^10

4. Address-Translation Scheme

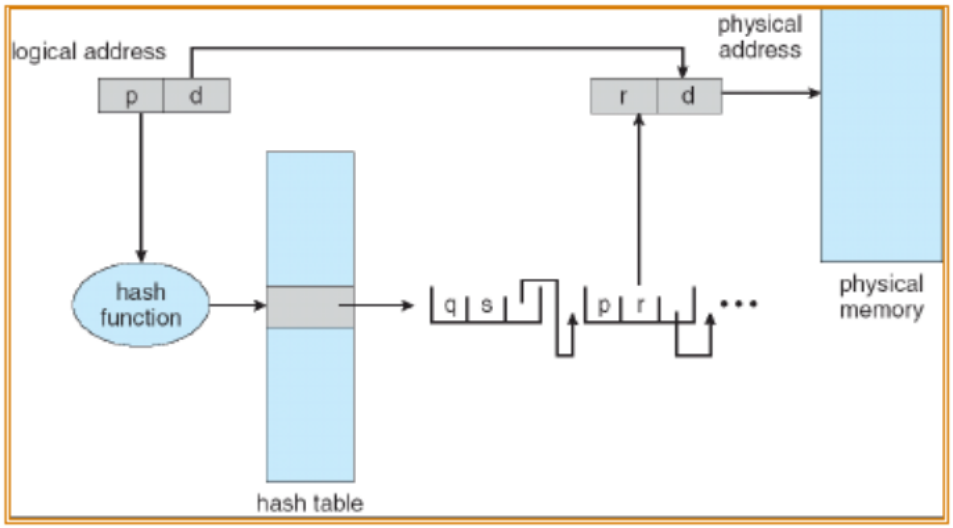
• Address-translation scheme for a two-level 32-bit paging architecture



5. Hashed Page Tables

• Common approach for handling address spaces larger than 32 bits
• The virtual page number is hashed into a page table.
This page table contains a chain of elements hashing to the same location
- Each element consists of three fields:
1) the virtual page number,
2) the value of the mapped page frame,
3) a pointer to the next element
• Virtual page numbers are compared in this chain searching for a match
- If a match is found, the corresponding physical frame is extracted.

-> 검색시간 줄어듦

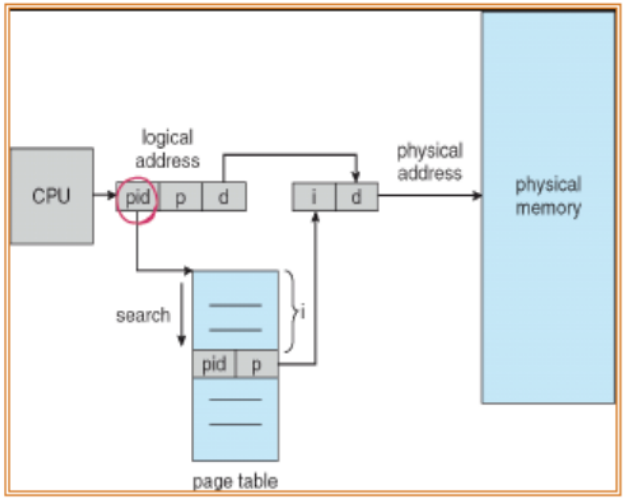


6. Inverted Page Table

• 논리적 공간 커질수록 페이지 테이블도 커지는거 해결, 테이블 엔트리 개수는 프레임 개수와 동일하다
• has one entry for each real page frame of memory
• Entry consists of
1) the page number stored in that real memory location, 물리적
2) the process ID that owns that page
• decreases memory needed to store each page table, but
• increases time needed to search the table when a page reference occurs

7. Inverted Page Table Architecture

• Inverted page table entry
- process id, page number
- The index is the corresponding frame number
• When a memory reference occurs, <pid, page number> is presented
• The inverted page table is searched for a match.
• If a match is found at entry i, then
<i, offset> is physical address
• If no match is found, then an illegal memory access



8. Shared Pages

• An advantage of paging is the code sharing

- Important in time-sharing environment.

• Shared code

- One copy of read-only code (reentrant code) shared among processes (i.e., text editors, compilers, window systems)

  • Reentrant code is non-self-modifying code, it never changes during execution

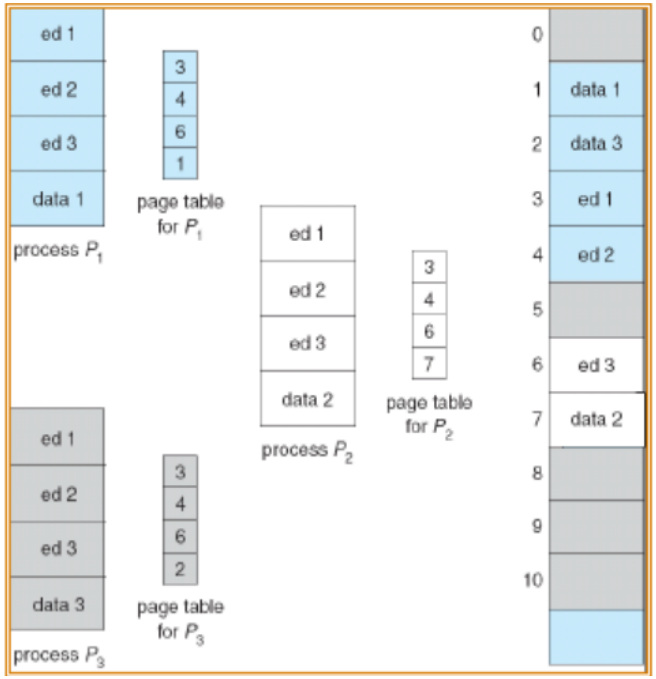
- Shared code must appear in same location in the logical address space of all processes

• Private code and data

- Each process keeps a separate copy of the code and data

- The pages for the private code and data can appear anywhere in the logical address space

9. Shared Pages Example



• 40 users, each of which executes a text editor

• Text editor:

- 150 KB of code,

- 50 KB of data

• Total memory space:

- 200 KB x 40 = 8000 KB

• If the code is reentrant code, it can be shared among 40 text editors.

- 150 KB for code is required.

• Total memory space

- 150 + 50 x 40 = 2150 KB

10. Segmentation

• User view of memory for a program

- A program is a collection of segments.

• A segment is a logical unit such as:

main program, procedure, function, method, object, local variables, global variables, common block, stack, symbol table, arrays

• Segmentation is a memory-management scheme that supports user view of memory

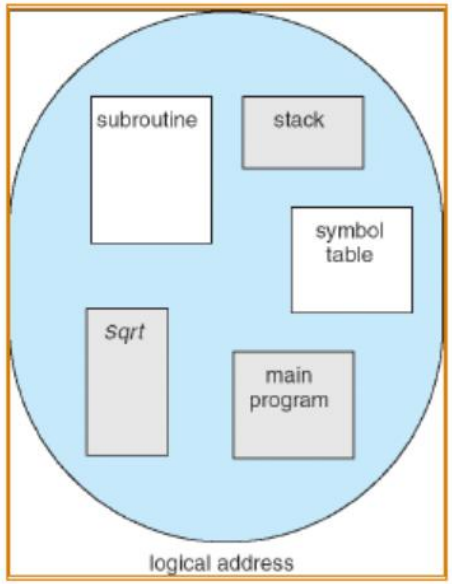
10-1. User's View of a Program

• A program is a collection of segments

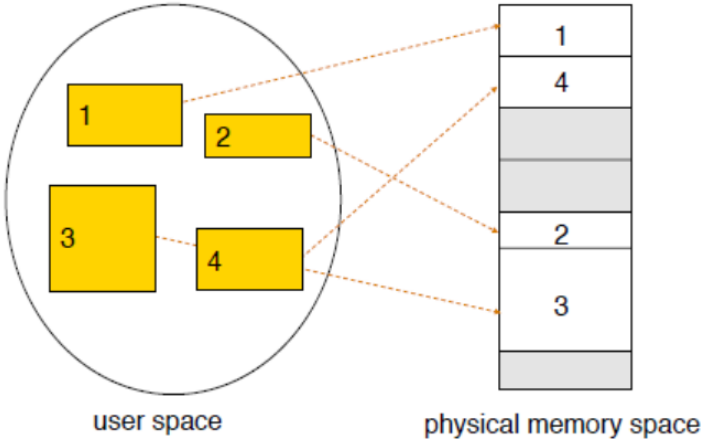
- main program
- subroutine
- stack
- symbol table

• Elements within a segment are identified by their offset from the beginning of the segment

- The 1st statement of the sqrt
- The 5th element in the stack.
- <segment-name, offset> for addressing



10-2. Logical View of Segmentation



• A logical address space is a collection of segments.
• Each segment has its name and a length
• <segment-name, offset> is used for addressing
• Segmentation maps each segment in the physical memory

10-3. Example

• Normally, the user program is compiled, and the compiler automatically construct segments reflecting the input program.
• A C compiler might create separate segments for:
  - The code
  - Global variables
  - The heap, form which memory is allocated
  - The stacks used by each thread
  - The standard C library
• Libraries might be assigned separate segments
• The loader take these segments and assign them segment number

10-4. Segmentation Architecture

• Each segment is assigned a number: segment-number
• Logical address consists of a two tuple:
  <segment-number, offset>
• Segment table – maps two-dimensional logical address to physical addresses;
• Each table entry has:
  1) base – contains the starting physical address where the segments reside in memory
  2) limit – specifies the length of the segment
• Segment-table base register (STBR) points to the segment table's location in memory
• Segment-table length register (STLR) indicates number of segments used by a program;
  segment number s is legal if s < STLR

10-5. Address Translation Architecture